- **Some more simple assembly language programs**
- Using 9S12 input and output ports
- Huang Sections 7.2 through 7.5
    - Using a subroutine to wait for and respond to an event
    - Using an input port to chech the state of DIP switches
    - Using an output port to control LEDs
    - An assembly language program to display a pattern on a set of LEDs

Exam 1

• You will be able to use all of the Motorola data manuals on the exam.
• No calculators will be allowed for the exam.
• Numbers
– Decimal to Hex (signed and unsigned)
– Hex to Decimal (signed and unsigned)
– Binary to Hex
– Hex to Binary
– Addition and subtraction of fixed-length hex numbers
– Overflow, Carry, Zero, Negative bits of CCR
• Programming Model
– Internal registers – A, B, (D = AB), X, Y, SP, PC, CCR
• Addressing Modes and Effective Addresses
– INH, IMM, DIR, EXT, REL, IDX (Not Indexed Indirect)
– How to determine effective address
• Instructions
– What they do - Core Users Guide
– What machine code is generated
– How many cycles to execute
– Effect on CCR
– Branch instructions – which to use with signed and which with unsigned
• Machine Code
– Reverse Assembly
• Stack and Stack Pointer
– What happens to stack and SP for instructions (e.g., PSHX, JSR)
– How the SP is used in getting to and leaving subroutines
• Assembly Language
– Be able to read and write simple assembly language program
– Know basic assembler directives – e.g., equ, dc.b, ds.w
    – Flow charts

```
; Subroutine to wait for 100 ms

delay:      psha                ; 2 cycles
            pshx                ; 2 cycles
            ldaa    #250        ; 1 cycle
loop2:      ldx     #3200       ; 2 cycles ------------
loop1:      dbne    x,loop1     ; 3 cycles inner loop | Outer loop
            dbne    a,loop2     ; 3 cycles ------------
            pulx                ; 3 cycls
            pula                ; 3 cycls
            rts                 ; 5 cycls
```

• Inner loop takes 3 cycles; is executed 3200 (X) times
• Outer loop takes (2 + 3X + 3) cycles; is executed 250 (A) times
• Total number of cycles: 2+2+1+A*(2+3X+3)+3+3+5 = 2,401,266 cycles
• This takes 100 ms with 24 MHz E-clock

Programming the HC12 in C

• A comparison of some assembly language and C constructs

```
Assembly                                    |           C
--------------------------------------------|-----------------------------------------------
; Use a name instead of a num               |   /* Use a name instead of a num */
COUNT: EQU 5                                |   #define COUNT 5

;-------------------------------------------|   /*----------------------------*/
;start a program                            |   /* To start a program */
        org     $1000                       |   main()
        lds     #0x3C00                     |   {
                                            |   }
;-------------------------------------------|   /*----------------------------*/
```

• Note that in C, the starting location of the program is defined when you compile the program, not in the program itself.
• Note that C always uses the stack, so C automatically loads the stack pointer for you.

```
Assembly                                    |           C
--------------------------------------------|-----------------------------------------------
;allocate two bytes for                     |   /* Allocate two bytes for
;a signed number                            |   * a signed number */
                                            |
        org     $2000                       |
i:      ds.w    1                           |   int i;
j:      dc.w    $1A00                       |   int j = 0x1a00;
;-------------------------------------------|   /*----------------------------*/
```

```
;allocate two bytes for                  |       /* Allocate two bytes for
;an unsigned number                      |       * an unsigned number */
i:      ds.w    1                       |       unsigned int i;
j:      dc.w    $1A00                   |       unsigned int j = 0x1a00;
;----------------------------------------|       /*---------------------------*/
;allocate one byte for                   |       /* Allocate one byte for
;an signed number                        |       * an signed number */
|
i:      ds.b    1                       |       signed char i;
j:      dc.b    $1F                     |       signed char j = 0x1f;

Assembly                                |               C
;----------------------------------------|       /*---------------------------*/
;Get a value from an address             |       /* Get a value from an address */
; Put contents of address                |       /* Put contents of address */
; $E000 into variable i                  |       /* 0xE000 into variable i */
                                         |
i:      ds.b    1                       |       unsigned char i;
                                         |
        ldaa    $E000                   |       i = * (unsigned char *) 0xE000;
        staa    I                       |
                                         |       /*---------------------------------*/
                                         |       /* Use a variable as a pointer
                                         |       (address) */
                                         |
                                         |       unsigned char *ptr, i;
                                         |
                                         |       ptr = (unsigned char *) 0xE000;
                                         |       i = *ptr;
                                         |       *ptr = 0x55;
;----------------------------------------|       /*---------------------------*/
```

• In C, the construct *(num) says to treat num as an address, and to work with the contents of that address.
• Because C does not know how many bytes from that address you want to work with, you need to tell C how many bytes you want to work with. You also have to tell C whether you want to treat the data as signed or unsigned.
– i = * (unsigned char *) 0xE000; tells C to take one byte from address 0xE000, treat it as unsigned, and store that value in variable i.
– j = * (int *) 0xE000; tells C to take two bytes from address 0xE000, treat it as signed, and store that value in variable j.
– * (char *) 0xE000 = 0xaa; tells C to write the number 0xaa to a single byte at addess 0xE000.
– * (int *) 0xE000 = 0xaa; tells C to write the number 0x00aa to two bytes starting at addess 0xE000.

Assembly	C
;----------	/*----------*/
;To call a subroutine	/* To call a function */
ldaa I	sqrt(i);
jsr sqrt	
;----------	/*----------*/
;To return from a subroutine	/* To return from a function */
ldaa j	return j;
rts	
;----------	/*----------*/
;Flow control	/* Flow control */
blo	if (i < j)
blt	if (i < j)
bhs	if (i >= j)
bge	if (i >= j)
;----------	/*----------*/

• Here is a simple program written in C and assembly. It simply divides 16 by 2. It does the division in a function.

ASSEMBLY	C
org $2000	signed char i;
i: ds.b 1	
	signed char div(signed char j);
org $1000	main()
lds #$3C00	{
ldaa #16	i = div(16);
jsr div	}
staa I	
swi	
div: asra	signed char div(signed char j)
rts	{
	return j >> 1;
	}

A simple C program and how to compile it

Here is a simple C program

```
#define COUNT 5
unsigned int i;
main()
{
        i = COUNT;
}
```

Details of compiling a program are discussed in detail in the text in Section 5.10. Here is an outline of the details:

1. Open the Embedded GNU (EGNU) IDE.

2. From the File menu, select the New Source File option. Type in your C program. Then from the File menu, select the Save unit as submenu, and save your file with an appropriate name and in an appropriate directory.

3. From the *File* menu, select the *New Project* option. Give the project an appropriate name and an appropriate directory. (Note: the project base name must be different from the C file names.) When the *Project Options* popup dialog appears, click the down arrow below *Hardware Profile*, and select *Dragon12*. Click the *Edit Profile* button, and make sure the following are set:

- ioports from 0000, length 400
- eeprom from 400, length c00
- data from 1000, length 1000
- text from 2000, length 2000
- stack at 3c00

Then click the OK button.

4. From the Project menu, select the Add to project option, and, in the pop-up dialog box, select the C file you entered in Step 2.

5. From the Build menu, select the Make option. Under the Compiler window at the bottom of the screen, you will hopefully see the message No errors or warnings. If not, you will need to fix the errors.

6. If all went well, you should be able to download the compiled file into the 9S12.

If the name of your project is ***Project1.prj***, the compiler will generate a file ***Project1.dmp***. Here is some of the output from ***Project1.dmp***. There are a couple of things you should note about this output:

• The first thing the C program does is load the stack pointer.

• The main() function is the assembly language for the C program you entered.

```
Disassembly of section .text:

00002000 <_start>:
    2000:   cf 3c 00          lds   #3c00 <_stack>
    2003:   16 20 38          jsr   2038 <__premain>

00002006 <__map_data_section>:
    2006:   ce 20 42          ldx   #2042 <__data_image>
    2009:   cd 10 00          ldy   #1000 <__data_section_start>
    200c:   cc 00 00          ldd   #0 <__data_section_size>
    200f:   27 07             beq   2018 <__init_bss_section>

00002011 <Loop>:
    2011:   18 0a 30 70       movb  1,X+, 1,Y+
    2015:   04 34 f9          dbne  D,2011 <Loop>

00002018 <__init_bss_section>:
    2018:   cc 00 02          ldd   #2 <__bss_size>
    201b:   27 08             beq   2025 <Done>
    201d:   ce 10 00          ldx   #1000 <__data_section_start>

00002020 <Loop>:
    2020:   69 30             clr   1,X+
    2022:   04 34 fb          dbne  D,2020 <Loop>

00002025 <Done>:
    2025:   16 20 31          jsr   2031 <main>

00002028 <fatal>:
    2028:   16 20 3c          jsr   203c <_exit>
    202b:   20 fb             bra   2028 <fatal>
    202d:   20 06             bra   2035 <main+0x4>
    202f:   20 18             bra   2049 <__data_image+0x7>

00002031 <main>:
    2031:   18 03 00 05       movw  #5 <__bss_size+0x3>, 1000
<__data_section_start>
    2035:   10 00
    2037:   3d                rts

00002038 <__premain>:
    2038:   87                clra
    2039:   b7 02             tap
    203b:   3d                rts

0000203c <_exit>:
    203c:   10 ef             cli
    203e:   3e                wai
    203f:   20 fb             bra   203c <_exit>

00002041 <_etext>:
```

```
2041:   a7              nop
```

Pointers in C

• To access a memory location:

```
*address
```

• You need to tell compiler whether you want to access 8-bit or 16 bit number, signed or unsigned:

```
*(type *)address
```

– To read from an eight-bit unsigned number at memory location 0x2000:

```
x = *(unsigned char *)0x2000;
```

– To write an 0xaa55 to a sixteen-bit signed number at memory locations 0x2010 and 0x2011:

```
*(signed int *)0x2010 = 0xaa55;
```

• If there is an address which is used a lot:

```
#define PORTA (* (unsigned char *) 0x0000)
x = PORTA; /* Read from address 0x0000 */
PORTA = 0x55; /* Write to address 0x0000 */
```

• To access consecutive locations in memory, use a variable as a pointer:

```
unsigned char *ptr;
ptr = (unsigned char *)0x2000;
*ptr = 0xaa; /* Put 0xaa into address 0x2000 */
ptr = ptr+2; /* Point two further into table */
x = *ptr; /* Read from address 0x2002 */
```

• To set aside ten locations for a table:

```
unsigned char table[10];
```

• Can access the third element in the table as:

```
table[2]
```

or as

```
*(table+2)
```

• To set up a table of constant data:

```
const unsigned char table[] = {0x00,0x01,0x03,0x07,0x0f};
```

This will tell the compiler to place the table of constant data with the program (which might be placed in EEPROM) instead of with regular data (which must be placed in RAM).

• There are a lot of registers (such as PORTA and DDRA) which you will use when programming in C. Rather than having to define the registers each time you use them, you can include a header file for the HC12 which has the registers predefined. Here is a sample of the hcs12.h. You can find the complete file on the EE 308 homepage.

Here are a few entries from the header file:

```
#define     IOREGS_BASE  0x0000

#define     _IO8(off)    *(unsigned char  volatile *)(IOREGS_BASE + off)
#define     _IO16(off)   *(unsigned short volatile *)(IOREGS_BASE + off)

#define     PORTA        _IO8(0x00)     //port a = address lines a8 - a15
#define     PORTB        _IO8(0x01)     //port b = address lines a0 - a7
#define     DDRA         _IO8(0x02)     //port a direction register
#define     DDRB         _IO8(0x03)     //port a direction register

#define     PORTE        _IO8(0x08)     //port e = mode, irq and control signals
#define     DDRE         _IO8(0x09)     //port e direction register
#define     PEAR         _IO8(0x0A)     //port e assignments
#define     MODE         _IO8(0x0B)     //mode register
#define     PUCR         _IO8(0x0C)     //port pull-up control register
#define     RDRIV        _IO8(0x0D)     //port reduced drive control register
#define     EBICTL       _IO8(0x0E)     //stretch control
```